To determine which sequences could not occur when performing push and pop operations on a stack, we need to simulate the stack behavior. A stack is a Last-In-First-Out (LIFO) data structure, meaning the last element pushed onto the stack is the first one to be popped off.

To determine when to push and when to pop while verifying a sequence of stack operations, we need to simulate the stack behavior based on the desired output sequence. Here's a step-by-step explanation of how this is done:

Approach to Determine Push and Pop Operations

1. **Initial Setup:**
    - Start with an empty stack and a list of numbers to be pushed: `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`.
    - Initialize an index or pointer to track the current position in the sequence we are trying to match.
2. **Simulate Operations:**
    - **Push Operation:** Push the next number from the list onto the stack.
    - **Pop Operation:** If the number on top of the stack matches the current number in the output sequence, pop it from the stack.
3. **Iterate Through the Sequence:**
    - For each number in the sequence:
        - If the stack's top does not match the current number in the sequence, push numbers from the list onto the stack until it matches or you run out of numbers.
        - Pop the stack if the top matches the current number in the sequence.
4. **Continue Until the Sequence is Completed:**
    - Repeat the process until you either complete the sequence or reach a point where the stack's top cannot match the next number in the sequence (making the sequence impossible).

Example for Sequence b: 4 6 8 7 5 3 2 9 0 1

Let's walk through the sequence step by step to see when we push and pop: The target number is selected by following the order of numbers in the given output sequence. The goal is to determine whether it's possible to produce this sequence using a series of valid push and pop operations on a stack.

1. **Push 0, 1, 2, 3, 4.**
    - Stack: `4, 3, 2, 1, 0` (top to bottom)

- Sequence target: `4`
- **Pop 4.**
  - Stack after pop: `3, 2, 1, 0`

2. **Sequence target:** `6`

- **Push 5, 6.**
  - Stack: `6, 5, 3, 2, 1, 0`
- **Pop 6.**
  - Stack after pop: `5, 3, 2, 1, 0`

3. **Sequence target:** `8`

- **Push 7, 8.**
  - Stack: `8, 7, 5, 3, 2, 1, 0`
- **Pop 8.**
  - Stack after pop: `7, 5, 3, 2, 1, 0`

4. **Sequence target:** `7`

- **Pop 7.**
  - Stack after pop: `5, 3, 2, 1, 0`

5. **Sequence target:** `5`

- **Pop 5.**
  - Stack after pop: `3, 2, 1, 0`

6. **Sequence target:** `3`

- **Pop 3.**
  - Stack after pop: `2, 1, 0`

7. **Sequence target:** `2`

- **Pop 2.**
  - Stack after pop: `1, 0`

8. **Sequence target:** `9`

- 9 was never pushed onto the stack after the previous pops, making this impossible. At this point, the stack only has `1, 0`, so popping 9 is not possible.

Conclusion

The key is to push numbers onto the stack only when necessary to match the next target number in the sequence, ensuring the stack operates in a LIFO manner. The process is iterated to verify if it's possible to create the sequence with valid push and pop operations. If at any point a number in the sequence cannot be matched by popping the stack, the sequence is deemed impossible. Let's evaluate each sequence to determine whether it can be generated by a series of push and pop operations on a stack:

1. **Sequence a: 4 3 2 1 0 9 8 7 6 5**

- Push 0, 1, 2, 3, 4 onto the stack.
- Pop 4, 3, 2, 1, 0 from the stack.
- Push 5, 6, 7, 8, 9 onto the stack.
- Pop 9, 8, 7, 6, 5 from the stack.

This sequence is possible.

2. **Sequence b: 4 6 8 7 5 3 2 9 0 1**

- Push 0, 1, 2, 3, 4 onto the stack.
- Pop 4 from the stack.
- Push 5, 6 onto the stack.
- Pop 6 from the stack.
- Push 7, 8 onto the stack.
- Pop 8, 7 from the stack.
- Pop 5 from the stack.
- Pop 3, 2 from the stack (The stack after the last operation looks like 9, 1, 0. However, 9 was never pushed onto the stack during this sequence, making it impossible to pop 9 at this point. The sequence violates the stack order because 9 was not added to the stack after popping 2).

This sequence is not possible.

3. **Sequence c: 2 5 6 7 4 8 9 3 1 0**

- Push 0, 1, 2 onto the stack.
- Pop 2 from the stack.
- Push 3, 4, 5, 6, 7 onto the stack.
- Pop 5, 6, 7 from the stack.
- Pop 4 from the stack.
- Push 8, 9 onto the stack.
- Pop 8, 9 from the stack.
- Pop 3, 1, 0 from the stack.

This sequence is possible.

4. **Sequence d: 4 3 2 1 0 5 6 7 8 9**

- Push 0, 1, 2, 3, 4 onto the stack.
- Pop 4, 3, 2, 1, 0 from the stack.
- Push 5, 6, 7, 8, 9 onto the stack.
- Pop 5, 6, 7, 8, 9 from the stack.

This sequence is possible.

5. **Sequence e: 1 2 3 4 5 6 9 8 7 0**

- Push 0, 1 onto the stack.
- Pop 1 from the stack.
- Push 2 onto the stack.

- Pop 2 from the stack.
- Push 3 onto the stack.
- Pop 3 from the stack.
- Push 4 onto the stack.
- Pop 4 from the stack.
- Push 5, 6 onto the stack.
- Pop 5, 6 from the stack.
- Push 7, 8, 9 onto the stack.
- Pop 9, 8, 7 from the stack.
- Pop 0 from the stack (not possible as 0 was at the bottom when 9 was at the top).

This sequence is not possible.

6. **Sequence f: 0 4 6 5 3 8 1 7 2 9**

- Push 0 onto the stack.
- Pop 0 from the stack.
- Push 1, 2, 3, 4 onto the stack.
- Pop 4 from the stack.
- Push 5, 6 onto the stack.
- Pop 6, 5 from the stack.
- Pop 3 from the stack.
- Push 7, 8 onto the stack.
- Pop 8 from the stack.
- Pop 1 from the stack (not possible as 1 is below 3 in the stack when 3 is popped).

This sequence is not possible.

7. **Sequence g: 1 4 7 9 8 6 5 3 0 2**

- Push 0, 1 onto the stack.
- Pop 1 from the stack.
- Push 2, 3, 4 onto the stack.
- Pop 4 from the stack.
- Push 5, 6, 7 onto the stack.
- Pop 7 from the stack.
- Push 8, 9 onto the stack.
- Pop 9, 8 from the stack.
- Pop 6, 5 from the stack.
- Pop 3 from the stack.
- Pop 0 from the stack (not possible as 0 is below 3 in the stack when 3 is popped).

This sequence is not possible.

8. **Sequence h: 2 1 4 3 6 5 8 7 9 0**

- Push 0, 1, 2 onto the stack.
- Pop 2 from the stack.

- Pop 1 from the stack.
- Push 3, 4 onto the stack.
- Pop 4 from the stack.
- Pop 3 from the stack.
- Push 5, 6 onto the stack.
- Pop 6, 5 from the stack.
- Push 7, 8, 9 onto the stack.
- Pop 8, 7, 9 from the stack.
- Pop 0 from the stack (not possible as 0 is below 9 in the stack when 9 is popped).

This sequence is not possible.

Conclusion

The sequences that **cannot** occur are:

- b. 4 6 8 7 5 3 2 9 0 1
- e. 1 2 3 4 5 6 9 8 7 0
- f. 0 4 6 5 3 8 1 7 2 9
- g. 1 4 7 9 8 6 5 3 0 2
- h. 2 1 4 3 6 5 8 7 9 0